

This Page Is Inserted by IFW Operations
and is not a part of the Official Record

BEST AVAILABLE IMAGES

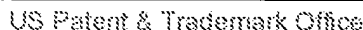
Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

**As rescanning documents *will not* correct images,
please do not report the images to the
Image Problem Mailbox.**




Search:  The ACM Digital Library  The Guide



THE ACM DIGITAL LIBRARY



 [Feedback](#) [Report a problem](#) [Satisfaction survey](#)

Found 120 of 120

relevance



☐ Open results in a new window

Try this search in The ACM Guide

expanded form

Result page: **1** 2 3 4 5 6 7 next

Relevance scale

-

March 1978 **ACM SIGMOD Record**, Volume 8 Issue 2Full text available: pdf(1.03 MB)

Additional Information: full citation, abstract, references

The presence of pointer variables in high level programming languages constitutes an artifact originally introduced to support the representation of recursive data structures. Programming practice has come to rely on pointer variables for their originally intended use, and for several others as well. Their use adds to the complexity of stating algorithms by forcing one to conceptualize data representations in which storage addressing is made manifest. In addition, the use of pointer variables al ...

- 

January 1987 **Computational Linguistics**, Volume 13 Issue 1-2

Full text available:

pdf(6.15 MB)

Additional Information: full citation

Publisher Site

-

F. L. Bauer, M. Broy, R. Gnatz, W. Hesse, B. Krieg-Brückner, H. Partsch, P. Pepper, H. Wössner
December 1978 **ACM SIGPLAN Notices**, Volume 13 Issue 12

Full text available: pdf(720.47 KB)

Additional Information: [full citation](#), [references](#), [citations](#)

-

January 1988 **Proceedings of the 15th ACM SIGPLAN-SIGACT symposium on Principles of programming languages**

Full text available: pdf (1.08 MB)

Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)

The choice of binding time disciplines has major consequences for both the run-time efficiency of programs and the convenience of the language expressing algorithms. Late

[Web](#) [Images](#) [Groups](#) [News](#) [Froogle](#)^{New!} [more »](#)[Advanced Search](#)
[Preferences](#)**Web**Results 1 - 3 of 3 for **interprocedural analysis partial order**. (0.17 seconds)**[PS] [Polyvariance, Polymorphism and Flow Analysis](#)**File Format: Adobe PostScript - [View as Text](#)... The key to the usefulness of the **anaysis** is the ... A flexible approach to **interprocedural** data flow. ... Symposium on **Partial** Evaluation and Semantics-Based Program ...www.it.kth.se/~kff/tomaps.ps - [Similar pages](#)**[PS] [To appear in Scientific Programming, 1999](#)**File Format: Adobe PostScript - [View as Text](#)... attempts to **partially** perform data-flow analysis at run time, using ... **Interprocedural** conditional branch elim-. ... Systematic design of program **anaysis** frameworks. ...www.isi.edu/~bsc/sp99.ps - [Similar pages](#)**[PS] [Abstract Interpretation: a Semantics-Based Tool for Program](#)**File Format: Adobe PostScript - [View as Text](#)... A simple way is to equip E with a **partial order** v, where xvy intuitively means "x is a more precise description than y", eg + v ?. ...www.mpi-sb.mpg.de/~podelski/Teaching/SS02/JonesNielson.ps - Supplemental Result - [Similar pages](#)[Search within results](#) | [Language Tools](#) | [Search Tips](#) | [Dissatisfied? Help us improve](#)[Google Home](#) - [Advertising Programs](#) - [Business Solutions](#) - [About Google](#)

©2004 Google


[Subscribe \(Full Service\)](#) [Register \(Limited Service, Free\)](#) [Login](#)

 Search: ☒ The ACM Digital Library ☐ The Guide

inter*procedur* <near> lattice and partial <near> order*



THE ACM DIGITAL LIBRARY


[Feedback](#) [Report a problem](#) [Satisfaction survey](#)

 Terms used **inter** **procedur** **near** **lattice** and **partial** **near** **order**

Found 30,799 of 134,837

Sort results by

relevance

Display results

expanded form

☒ Save results to a Binder

☒ Search Tips

☐ Open results in a new window

[Try an Advanced Search](#)
[Try this search in The ACM Guide](#)

Results 1 - 20 of 200

 Result page: [1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [next](#)

Best 200 shown

 Relevance scale ☐ ☐ ☐ ☐ ☐

1 [A framework for call graph construction algorithms](#)

David Grove, Craig Chambers

 November 2001 **ACM Transactions on Programming Languages and Systems (TOPLAS)**, Volume 23 Issue 6

Full text available: pdf(1.36 MB)

 Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)

A large number of call graph construction algorithms for object-oriented and functional languages have been proposed, each embodying different tradeoffs between analysis cost and call graph precision. In this article we present a unifying framework for understanding call graph construction algorithms and an empirical comparison of a representative set of algorithms. We first present a general parameterized algorithm that encompasses many well-known and novel call graph construction algorithms. W ...

Keywords: Call graph construction, control flow analysis, interprocedural analysis

2 [Call graph construction in object-oriented languages](#)

David Grove, Greg DeFouw, Jeffrey Dean, Craig Chambers

 October 1997 **ACM SIGPLAN Notices , Proceedings of the 12th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications**, Volume 32 Issue 10

Full text available: pdf(2.63 MB)

 Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)

Interprocedural analyses enable optimizing compilers to more precisely model the effects of non-inlined procedure calls, potentially resulting in substantial increases in application performance. Applying interprocedural analysis to programs written in object-oriented or functional languages is complicated by the difficulty of constructing an accurate program call graph. This paper presents a parameterized algorithmic framework for call graph construction in the presence of message sends and/or ...

3 [Complete removal of redundant expressions](#)

Rastislav Bodík, Rajiv Gupta, Mary Lou Soffa

 May 1998 **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 1998 conference on Programming language design and implementation**, Volume 33 Issue 5

Full text available: pdf(2.13 MB)

 Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)

Partial redundancy elimination (PRE), the most important component of global optimizers, generalizes the removal of common subexpressions and loop-invariant computations. Because existing PRE implementations are based on *code motion*, they fail to completely remove the redundancies. In fact, we observed that 73% of loop-invariant statements cannot be eliminated from loops by code motion alone. In dynamic terms, traditional PRE eliminates only half of redundancies that are strictly partial. ...

Keywords: control flow restructuring, demand-driven frequency data-flow analysis, partial redundancy elimination, profile-guided optimization, speculative execution

4 [Interprocedural static analysis of sequencing constraints](#)

Kurt M. Olender, Leon J. Osterweil

January 1992 **ACM Transactions on Software Engineering and Methodology (TOSEM)**,
Volume 1 Issue 1

Full text available:  [pdf\(2.37 MB\)](#)

Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#), [review](#)

This paper describes a system that automatically performs static interprocedural sequencing analysis from programmable constraint specifications. We describe the algorithms used for interprocedural analysis, relate the problems arising from the analysis of real-world programs, and show how these difficulties were overcome. Finally, we sketch the architecture of our prototype analysis system (called Cesar) and describe our experiences to date with its use, citing performance and error detect ...

Keywords: error detection, interprocedural data flow analysis, sequencing constraints

5 [Understanding class hierarchies using concept analysis](#)

Gregor Snelting, Frank Tip

May 2000 **ACM Transactions on Programming Languages and Systems (TOPLAS)**,
Volume 22 Issue 3

Full text available:  [pdf\(433.91 KB\)](#)

Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)


A new method is presented for analyzing and reengineering class hierarchies. In our approach, a class hierarchy is processed along with a set of applications that use it, and a fine-grained analysis of the access and subtype relationships between objects, variables, and class members is performed. The result of this analysis is again a class hierarchy, which is guaranteed to be behaviorally equivalent to the original hierarchy, but in which each object only contains the members that are req ...

Keywords: class hierarchy reengineering, concept analysis

6 [Fast interprocedural class analysis](#)

Greg DeFouw, David Grove, Craig Chambers

January 1998 **Proceedings of the 25th ACM SIGPLAN-SIGACT symposium on Principles of programming languages**


Full text available:  [pdf\(2.03 MB\)](#)

Additional Information: [full citation](#), [references](#), [citations](#), [index terms](#)

7 [Vectorization and parallelization of irregular problems via graph coloring](#)

Hans-Christian Hege, Hinnerk Stüben

June 1991 **Proceedings of the 5th international conference on Supercomputing**


Full text available:  [pdf\(1.03 MB\)](#)

Additional Information: [full citation](#), [references](#), [index terms](#)

8 [Reengineering class hierarchies using concept analysis](#)

Gregor Snelting, Frank Tip

November 1998 **ACM SIGSOFT Software Engineering Notes , Proceedings of the 6th ACM SIGSOFT international symposium on Foundations of software engineering**, Volume 23 Issue 6

Full text available:  [pdf\(1.31 MB\)](#)

Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)

The design of a class hierarchy may be imperfect. For example, a class *C* may contain a member *m* not accessed in any *C*-instance, an indication that *m* could be eliminated, or moved into a derived class. Furthermore, different subsets of *C*'s members may be accessed from different *C*-instances, indicating that it might be appropriate to split *C* into multiple classes. We present a framework for detecting and remediating such design problems, which is ba ...

9 [Automatic data layout for distributed-memory machines](#)

Ken Kennedy, Ulrich Kremer

July 1998 **ACM Transactions on Programming Languages and Systems (TOPLAS)**, Volume 20 Issue 4

Full text available:  [pdf\(633.20 KB\)](#)

Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#), [review](#)

The goal of languages like Fortran D or High Performance Fortran (HPF) is to provide a simple yet efficient machine-independent parallel programming model. After the algorithm selection, the data layout choice is the key intellectual challenge in writing an efficient program in such languages. The performance of a data layout depends on the target compilation system, the target machine, the problem size, and the number of available processors. This makes the choice of a good layout extremel ...

Keywords: high performance Fortran

10 [Array abstractions using semantic analysis of trapezoid congruences](#)

François Masdupuy

August 1992 **Proceedings of the 6th international conference on Supercomputing**

Full text available:  [pdf\(800.51 KB\)](#)


Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)

With the growing use of vector supercomputers, efficient and accurate data structure analyses are needed. What we propose in this paper is to use the quite general framework of Cousot's abstract interpretation for the particular analysis of multi-dimensional array indexes. While such indexes are integer tuples, a relational integer analysis is first required. This analysis results of a combination of existing ones that are interval and congruence based. Two orthogonal problems are directly ...

11 [Pointer analysis for multithreaded programs](#)

Radu Rugina, Martin Rinard

May 1999 **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 1999 conference on Programming language design and implementation**, Volume 34 Issue 5

Full text available:  [pdf\(1.82 MB\)](#)

Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)

This paper presents a novel interprocedural, flow-sensitive, and context-sensitive pointer analysis algorithm for multithreaded programs that may concurrently update shared pointers. For each pointer and each program point, the algorithm computes a conservative

approximation of the memory locations to which that pointer may point. The algorithm correctly handles a full range of constructs in multithreaded programs, including recursive functions, function pointers, structures, arrays, nested stru ...

12 ABCD: eliminating array bounds checks on demand

Rastislav Bodík, Rajiv Gupta, Vivek Sarkar

May 2000 **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 2000 conference on Programming language design and implementation**, Volume 35 Issue 5

Full text available:  pdf(306.96 KB)

Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)

To guarantee typesafe execution, Java and other strongly typed languages require bounds checking of array accesses. Because array-bounds checks may raise exceptions, they block code motion of instructions with side effects, thus preventing many useful code optimizations, such as partial redundancy elimination or instruction scheduling of memory operations. Furthermore, because it is not expressible at bytecode level, the elimination of bounds checks can only be performed at run time ...

13 Efficient computation of flow insensitive interprocedural summary information

Keith D. Cooper, Ken Kennedy

June 1984 **ACM SIGPLAN Notices , Proceedings of the 1984 SIGPLAN symposium on Compiler construction**, Volume 19 Issue 6


Full text available:  pdf(970.59 KB)

Additional Information: [full citation](#), [references](#), [citations](#)

14 Curriculum 68: Recommendations for academic programs in computer science: a report of the ACM curriculum committee on computer science

William F. Atchison, Samuel D. Conte, John W. Hamblen, Thomas E. Hull, Thomas A. Keenan, William B. Kehl, Edward J. McCluskey, Silvio O. Navarro, Werner C. Rheinboldt, Earl J. Scheppe, William Viavant, David M. Young

March 1968 **Communications of the ACM**, Volume 11 Issue 3

Full text available:  pdf(6.63 MB)


Additional Information: [full citation](#), [references](#), [citations](#)

Keywords: computer science academic programs, computer science bibliographies, computer science courses, computer science curriculum, computer science education, computer science graduate programs, computer science undergraduate programs

15 Parallel execution of prolog programs: a survey

Gopal Gupta, Enrico Pontelli, Khayri A.M. Ali, Mats Carlsson, Manuel V. Hermenegildo

July 2001 **ACM Transactions on Programming Languages and Systems (TOPLAS)**, Volume 23 Issue 4

Full text available:  pdf(1.95 MB)

Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)

Since the early days of logic programming, researchers in the field realized the potential for exploitation of parallelism present in the execution of logic programs. Their high-level nature, the presence of nondeterminism, and their referential transparency, among other characteristics, make logic programs interesting candidates for obtaining speedups through parallel execution. At the same time, the fact that the typical applications of logic programming frequently involve irregular computatio ...

Keywords: Automatic parallelization, constraint programming, logic programming, parallelism, prolog

16 Performing data flow analysis in parallel

Yong-fong Lee, Thomas J. Marlowe, Barbara G. Ryder

November 1990 **Proceedings of the 1990 ACM/IEEE conference on Supercomputing**Full text available:  pdf (1.26 MB) Additional Information: [full citation](#), [abstract](#), [references](#)

We have designed a *family of parallel data flow analysis algorithms* for execution on a message-passing MIMD architecture, based on general-purpose, hybrid data flow analysis algorithms [22]. We have exploited the natural task partitioning of the hybrid algorithms and have explored a *static mapping-dynamic scheduling* strategy. Alternative mapping-scheduling choices and refinements of the flow graph condensation utilized are discussed. Our parallel hybrid algorithm family is illustra ...

Keywords: Data flow analysis, hybrid algorithms, message-passing machines, parallel algorithms, partitioning, scheduling

17 Analyzing the communication topology of concurrent programs

Christopher Colby

June 1995 **Proceedings of the 1995 ACM SIGPLAN symposium on Partial evaluation and semantics-based program manipulation**Full text available:  pdf (1.11 MB) Additional Information: [full citation](#), [references](#), [citations](#), [index terms](#)**18** Intermediate representation engineering: Annotating Java libraries in support of whole-program optimization

Michael Thies

June 2002 **Proceedings of the inaugural conference on the Principles and Practice of programming, 2002 and Proceedings of the second workshop on Intermediate representation engineering for virtual machines, 2002**Full text available:  pdf (550.97 KB) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

Optimizing just-in-time compilation of Java programs depends on information gained from state-of-the-art program analysis techniques. To avoid extensive analysis at program execution time, analysis results for the available parts of a program can be precomputed at compile time and then combined at runtime. Program analyses operate on isolated program modules like libraries and annotate them with information that can be post-processed efficiently. We have applied this approach to several concrete ...

19 Case studies in embedded systems: The analysis and design of architecture systems for speech recognition on modern handheld-computing devices

Andreas Hagen, Daniel A. Connors, Bryan L. Pellom

October 2003 **Proceedings of the 1st IEEE/ACM/IFIP international conference on Hardware/software codesign & system synthesis**Full text available:  pdf (280.91 KB) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)


Growing demand for high performance in embedded systems is creating new opportunities to use speech recognition systems traditionally executed only on high performance systems. In several ways, the needs of embedded computing differ from those of more traditional general-purpose systems. Embedded systems have more stringent constraints on cost and power consumption that lead to design bottlenecks for many computationally-intensive applications. This paper characterizes the speech recognition pro ...

Keywords: embedded systems, performance, speech recognition

20 [Minimum cost interprocedural register allocation](#)

Steven M. Kurlander, Charles N. Fischer

January 1996 **Proceedings of the 23rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages**

Full text available:  [pdf \(1.23 MB\)](#)

Additional Information: [full citation](#), [references](#), [citations](#), [index terms](#)







Results 1 - 20 of 200

Result page: [1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [next](#)

The ACM Portal is published by the Association for Computing Machinery. Copyright © 2004 ACM, Inc.

[Terms of Usage](#) [Privacy Policy](#) [Code of Ethics](#) [Contact Us](#)

Useful downloads:  [Adobe Acrobat](#)  [QuickTime](#)  [Windows Media Player](#)  [Real Player](#)